RATHINAM TECHNICAL CAMPUS (AUTONOMOUS)

**Affiliated To Anna University, Chennai & Approved by AICTE, New Delhi.**

**Coimbatore, Tamil Nadu, India – 641 021**

Celebrate life

RATHINAM

# 23MC101 – EMBEDDED C PROGRAMMING LABORATORY

NAME :

BRANCH :

REGISTER NUMBER :

YEAR / SEMESTER :

ACADEMIC YEAR :

SUBJECT CODE :

SUBJECT NAME :

**Affiliated To Anna University, Chennai & Approved by AICTE, New Delhi.**

**Coimbatore, Tamil Nadu, India – 641 021**

# BONAFIDE CERTIFICATE

NAME :

ACADEMIC YEAR :

YEAR/SEMESTER :

BRANCH :

**UNIVERSITY REGISTER NUMBER:**

Certified that this is the Bonafide record of work done by the above student in the _____Laboratory during the year 2024-2025.

**Staff-in-Charge** **Head of the Department**

Submitted for the Practical Examination held on …….

**Internal Examiner**

# INDEX

| Sl. No. | DATE | EXPERIMENT NAME | PAGE NO. | MARKS | SIGN. |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

| Ex.no:1<br>Date: | C PROGRAM TO PRINT USER INPUT DATA |
|---|---|

## AIM

Write a c program to get a input data from the user.

## ALGORITHM

STEP-1: Start the program.

STEP-2: Include necessary header files.

STEP-3: Declare variables to store the user's name and age.

STEP-4: Prompt the user to input their name using printf.

STEP-5: Read the user's name using scanf.

STEP-6: Prompt the user to input their age using printf.

STEP-7: Read the user's age using scanf.

STEP-8: Display a greeting along with the user's name and age using printf.

STEP-9: End the program.

## PROGRAM

```
#include <stdio.h>
int main()
{
   char name[50];
   int age;
   printf("Please enter your name: ");
   scanf("%s", name);
   printf("Enter your age: ");
   scanf("%d", &age);
```

```
    printf("\nHello, %s! You are %d years old.\n", name, age);

  return 0;

}
```

**OUTPUT**


**RESULT**

| Ex.no: 2<br>Date: | C PROGRAM TO DO ARITHMETIC OPERATION |
|---|---|

## AIM

Write C program that performs basic arithmetic operations (+, -, *, /) on two numbers.

## ALGORITHM

STEP-1: Start the program.

STEP-2: Declare variables for two numbers (x and y), a variable to store the result, and a character variable for the operation.

STEP-3: Prompt the user to input the first number and store it in x.

STEP-4: Prompt the user to input the second number and store it in y.

STEP-5: Prompt the user to select an operation (+, -, *, /,%) and store it in the operation variable.

STEP-6: Use a switch-case statement to perform the operation based on the user's choice:

- Case +: Print the result of addition.
- Case -: Print the result of subtraction.
- Case *: Print the result of multiplication.
- Case /: Print the result of division.
- Case %: Print the result of modulo division.
- For an invalid operation, display an error message.

STEP-7: Use goto to jump back to the operator Input setup.

STEP- 8: Display the result.

STEP-9: End the program.

**PROGRAM**

```
#include<stdio.h>
int main() {

   int x, y;

   INPUT:

   printf("Enter two Number : ");
   scanf("%d%d", &x, &y);

   char op;
   printf("Enter the Operator [ +, -, *, /, %%] : ");
   scanf(" %c", &op);

   switch (op)
   {
   case '+':
      printf("Addition: %d + %d = %d \n", x, y, x+y);
      break;

   case '-':
      printf("Subtraction: %d - %d = %d \n", x, y, x-y);
      break;

   case '*':
      printf("Product: %d * %d = %d \n", x, y, x*y);
      break;
   case '/':
```

```
        printf("Division: %d / %d = %d \n", x, y, x/y);
        break;

    case '%':
        printf("Modulo Division: %d %% %d = %d \n", x, y, x%y);
        break;

    default:
        printf("ERROR: Invalid Operation, Please try again\n");

        goto INPUT;
        break;
    }

    return 0;
}
```

**OUTPUT**

```
monishak@intellect-192:~/Downloads/Embedc$ ./prog2
Enter two Number : 12 5
Enter the Operator [ +, -, *, /, %] : +
Addition: 12 + 5 = 17
monishak@intellect-192:~/Downloads/Embedc$ ./prog2
Enter two Number : 12 5
Enter the Operator [ +, -, *, /, %] : -
Subtraction: 12 - 5 = 7
monishak@intellect-192:~/Downloads/Embedc$ ./prog2
Enter two Number : 12 5
Enter the Operator [ +, -, *, /, %] : *
Product: 12 * 5 = 60
monishak@intellect-192:~/Downloads/Embedc$ ./prog2
Enter two Number : 12 5
Enter the Operator [ +, -, *, /, %] : /
Division: 12 / 5 = 2
monishak@intellect-192:~/Downloads/Embedc$ ./prog2
Enter two Number : 12 5
Enter the Operator [ +, -, *, /, %] : %
Modulo Division: 12 % 5 = 2
monishak@intellect-192:~/Downloads/Embedc$ ./prog2
Enter two Number : 12 5
Enter the Operator [ +, -, *, /, %] : &
ERROR: Invalid Operation, Please try again
Enter two Number :
```

**RESULT**

| Ex.no:3<br>Date: | C PROGRAM TO DO BITWISE OPERATIONS |
|---|---|

**AIM**

To write a C program using bitwise Operations.

**ALGORITHM**

STEP 1: Start the program.

STEP 2: Initialize the variable and values **a = 5 , b = 12.**

STEP 3: To Perform a Bitwise operations like

**AND -> &**

**OR -> |**

**NOT -> ^**

**XOR -> ~**

**LEFTSHIFT -> <<**

**RIGHTSHIFT -> <<**

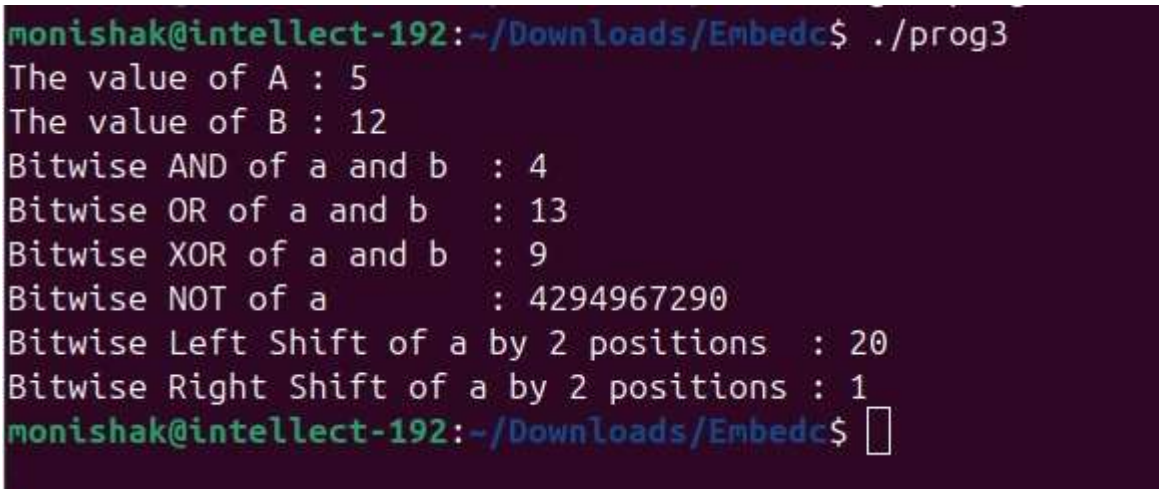
STEP 4: Print All “Operations”.

STEP 5: End the program.

**PROGRAM**

```
#include<stdio.h>
int main(){
   int a = 5, b = 12;
   printf("The value of A : %d\n", a);
   printf("The value of B : %d\n", b);
   printf("Bitwise AND of a and b  : %d\n", a & b);
   printf("Bitwise OR of a and b   : %u\n", a | b);
   printf("Bitwise XOR of a and b  : %u\n", a ^ b);
   printf("Bitwise NOT of a        : %u\n", ~a);
   printf("Bitwise Left Shift of a by 2 positions  : %u\n", a << 2);
   printf("Bitwise Right Shift of a by 2 positions : %u\n", a >> 2);
}
```

**OUTPUT**




**RESULT**

| Ex.no: 04<br>Date: | C PROGRAM TO FIND THE SIZE OF A GIVEN DATA TYPES |
|---|---|

## AIM

To write a C program find the Size of given data type.

## ALGORITHM

STEP 1: Start the program.

STEP 2: Initialize the all data types variables like

**char c;**

**int i;**

**float f;**

**double d;**

**long l;**

**short s;**

**long long ll;**

STEP 3: Print all data type size using **sizeof()** function

**sizeof(c)**

**sizeof(i)**

STEP 4: End the program.

## PROGRAM

```
#include <stdio.h>
void main() {
char c;
int i;
float f;
double d;
long l;
short s;
```

```
long long ll;
printf("Size of char       : %zu bytes\n", sizeof(c));
printf("Size of int        : %zu bytes\n", sizeof(i));
printf("Size of float      : %zu bytes\n", sizeof(f));
printf("Size of double     : %zu bytes\n", sizeof(d));
printf("Size of long       : %zu bytes\n", sizeof(l));
printf("Size of short      : %zu bytes\n", sizeof(s));
printf("Size of long long  : %zu bytes\n", sizeof(ll));
}
```

**OUTPUT**

```
monishak@intellect-192:~/Downloads/Embedc$ ./prog4
Size of char         : 1 bytes
Size of int          : 4 bytes
Size of float        : 4 bytes
Size of double       : 8 bytes
Size of long         : 8 bytes
Size of short        : 2 bytes
Size of long long    : 8 bytes
monishak@intellect-192:~/Downloads/Embedc$
```

**RESULT**

| Ex.no: 05<br>Date: | C PROGRAM TO IMPLEMENT LIBRARY FUNCTIONS USING DEFINED FUNCTIONS |
|---|---|

**AIM**

C program to implement library functions using user defined functions.

**ALGORITHM**

Step 1: Start the program.

Step 2: This function takes a pointer to a string (const char *str) as a parameter. It initializes a variable length to 0.

Step 3: It uses a while loop to iterate through each character of the string until the null character ('\0') is encountered.

Step 4: It increments the length for each character

Step 5: This function takes two parameters: char *dest (destination string) and const char *src (source string).

Step 6: It uses a while loop to copy each character from the source to the destination until the null character is encountered.

Step 7: A test string "Hello, World!" is used with the my_strlen function,andthe result is printed.

Step 8: A test string "Copy this!" is copied to a destination array using the my_strcpy function, and the result is printed.

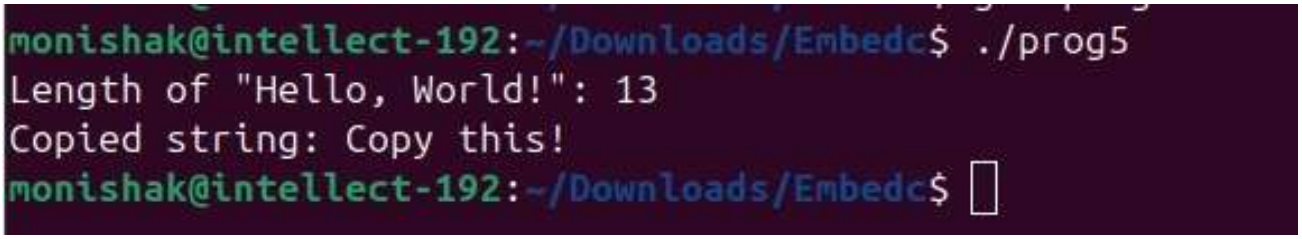
Step 9: Stop the program.

**PROGRAM**

```
#include <stdio.h>
int my_strlen(const char *str)
 {
   int length = 0;
   while (*str != '\0')
{
```

```
        length++;
        str++;
    }
    return length;
}
char* my_strcpy(char *dest, const char *src)
{
    char *original_dest = dest;
    while (*src != '\0')
{
        *dest = *src;
        dest++;
        src++;
    }
    *dest = '\0';
    return original_dest;
}
int main()
 {
    const char *testString = "Hello, World!";
    int length = my_strlen(testString);
    printf("Length of \"%s\": %d\n", testString, length);
    char destination[20];
    const char *source = "Copy this!";
    my_strcpy(destination, source);
    printf("Copied string: %s\n", destination);
    return 0;
}
```

**OUTPUT**




**RESULT**

| Ex.no: 06<br>Date: | C PROGRAM TO IMPLEMENT CONDITIONAL LOGICS |
|---|---|

**AIM**

C program to implement conditional logics.

**ALGORITHM**

Step 1: Start the program.

Step 2: Include the necessary header file for standard input and output functions.

Step 3: Define the main function.

Step 4: Declare an integer variable number to store user input.

Step 5: Prompt the user to enter a number and read the input.

Step 6: If the number is greater than 0, print that it's a positive number.

If the number is less than 0, print that it's a negative number.

Step 7: If neither condition is true, print that the number is zero.

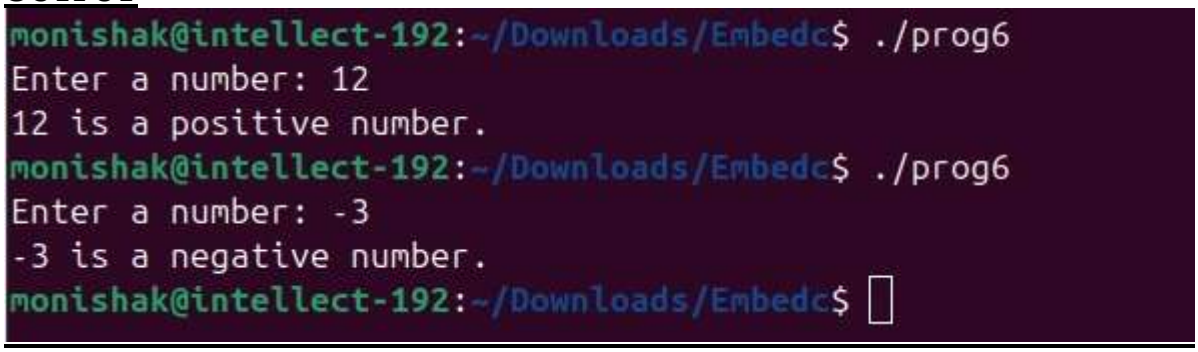
Step 9: Stop the program.

**PROGRAM**

```
#include <stdio.h>
int main()
 {
   int number;

   printf("Enter a number: ");
   scanf("%d", &number);

   if (number > 0)
{
      printf("%d is a positive number.\n", number);
   }
else if (number < 0)
```

```
{
        printf("%d is a negative number.\n", number);
    }
else
{
        printf("The number is zero.\n");
    }

    return 0;
}
```

**OUTPUT**




**RESULT**

| Ex.no: 07<br>Date: | C PROGRAM TO IMPLEMENT ITERATIVE LOGICS |
|---|---|

**AIM**

To write C program to implement iterative logics.

**ALGORITHM**

STEP-1: Accept the input value for which the factorial needs to be calculated.

STEP-2: Set a variable result to 1. This variable will store the factorial.

STEP-3: Use a for loop to iterate from 1 to the given number.

STEP-4: Multiply the current value of result by the loop variable in each iteration.

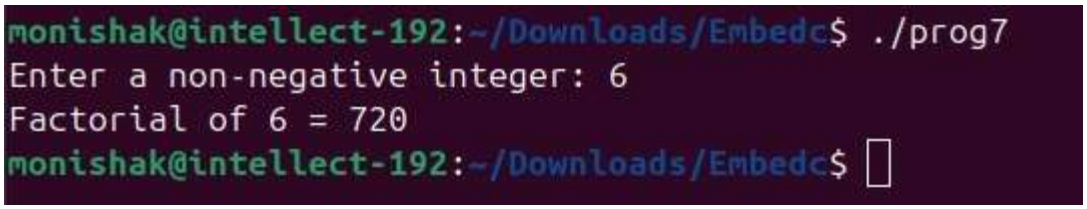
STEP-5: The final value of result is the factorial.

**PROGRAM**

```
#include <stdio.h>
unsigned long long factorial(int n)
 {
   unsigned long long result = 1;
   for (int i = 1; i <= n; ++i)
          {
      result *= i;
   }
   return result;
}
int main()
 {
   int num;
   printf("Enter a non-negative integer: ");
   scanf("%d", &num);
   if (num < 0)
```

```
        {
        printf("Factorial is not defined for negative numbers.\n");
    }
        else
        {
        unsigned long long result = factorial(num);
        printf("Factorial of %d = %llu\n", num, result);
    }
    return 0;
}
```

**Output**




**RESULT**

| Ex.no: 08<br>Date: | C PROGRAM TO IMPLEMENT CONDITIONAL COMPILATION |
|---|---|

## AIM

To write a C program to implement conditional compilation.

## ALGORITHM

STEP-1: include <stdio.h> includes the standard input/output library.

STEP-2: #define FEATURE_ENABLED defines a preprocessor macro named FEATURE_ENABLED. You can comment or uncomment this line to toggle the feature on or off.

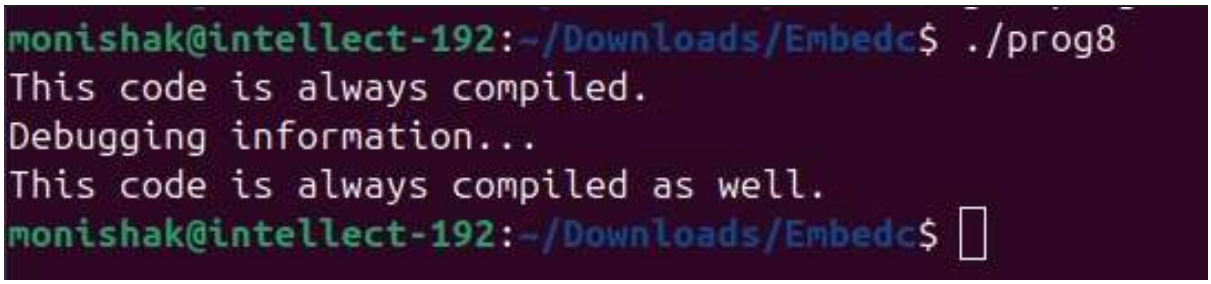
STEP-3: Inside the main function, #ifdef FEATURE_ENABLED checks if the FEATURE_ENABLED macro is defined. If defined, it prints "Feature is enabled"; otherwise, it prints "Feature is disabled."

STEP-4: The rest of the program follows after the conditional compilation.

## PROGRAM

```
#include <stdio.h>
#define DEBUG_MODE
int main() {
   printf("This code is always compiled.\n");
   #ifdef DEBUG_MODE
      printf("Debugging information...\n");
   #else
      printf("Release mode...\n");
   #endif
   printf("This code is always compiled as well.\n");
   return 0;
}
```

**OUTPUT**




**RESULT**

| Ex.no: 09<br>Date: | C PROGRAM TO SHOW RELATION BETWEEN POINTERS AND ARRAY |
|---|---|

## AIM

To write a c program to show relation between pointers and array.

## ALGORITHAM

STEP-1: Start the process.

STEP-2: Declare an array numbers containing integers {1, 2, 3, 4, 5}.

STEP-3: Declare an integer pointer ptr.

STEP-4: Point the pointer ptr to the first element of the array numbers.

STEP-5: Use array notation to print the value at index 2 of the array.

STEP-6: Use pointer notation to print the value at index 2 of the array.

STEP-7: Use a loop to iterate through the array elements using pointers and print each element.

STEP-8: Return 0 to indicate successful execution.

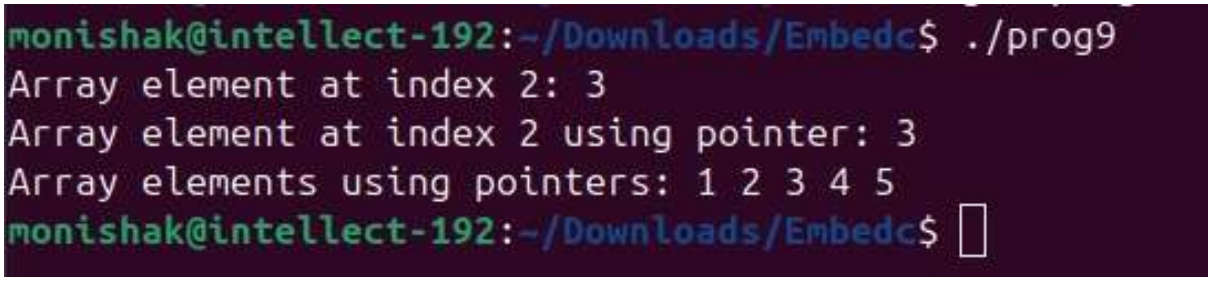
STEP-9: Stop the process.

## Program

```
#include <stdio.h>
 int main() {
    int numbers[] = {1, 2, 3, 4, 5};
    int *ptr;
   ptr = numbers;
   printf("Array element at index 2: %d\n", numbers[2]);
   printf("Array element at index 2 using pointer: %d\n", *(ptr + 2));
   printf("Array elements using pointers: ");
   for (int i = 0; i < 5; ++i) {
   printf("%d ", *(ptr + i)) ;
  }
  printf("\n");
return 0;
```

}

**OUTPUT**




**RESULT**

| Ex.no: 10<br>Date: | C PROGRAM TO IMPLEMANT LOGICS IN STRINGS |
|---|---|

**AIM**

To write a c program to implement logics in strings.

**ALGORITHM**

STEP 1: Start the process.

STEP 2: Use the strlen function to calculate the length of the string str1.

STEP 3: Print the calculated length.

STEP 4: Use the strcat function to concatenate str2 to the end of str1.

STEP 5: Print the concatenated string.

STEP 6: Use the strcmp function to compare str1 and str2.

STEP 7: Check if the result is equal to 0 (strings are equal) or not.

STEP 8: Print whether the strings are equal or not.

STEP 9: Calculate the length of the string str1.

STEP 10: Use a loop to swap characters from the beginning to the middle with their corresponding characters from the end.

STEP11: Print the reversed string.

STEP 12: Stop the processs,

**PROGRAM**

```
#include <stdio.h>
#include <string.h>
int main() {
    char str1[] = "Hello";
    char str2[] = " World!";
   printf("Length of str1: %lu\n", strlen(str1));
    strcat(str1, str2);
    printf("Concatenated string: %s\n", str1);
```

```
    if (strcmp(str1, str2) == 0)
{
        printf("Strings are equal\n");
    } else {
        printf("Strings are not equal\n");
    }
    int length = strlen(str1);
    for (int i = 0; i < length / 2; i++) {
        char temp = str1[i];
        str1[i] = str1[length - i - 1];
        str1[length - i - 1] = temp;
    }
    printf("Reversed string: %s\n", str1);
    return 0;
}
```

**OUTPUT**


**RESULT**

| Ex.no: 11<br>Date: | MAKEFILE TO COMPILE SINGLE & MULTIPLE FILES |
|---|---|

**AIM**

To make file to compile single & multiple files.

**ALGORITHM**

**Single File:**

STEP-1: Start the process.

STEP-2: Create a file to print the multiplication table for the user entered number.

STEP-3: Create a makefile for the file to print the multiplication table.

STEP-4: Enter the command to compile the file ,

makefile:multiplication_table.c

gcc multiplication_table.c -o multiplea

STEP-5: Enter the command to run the file,

run:

./multiplea

Step-6: Enter the command to remove the output file,

clean:

rm multiplea

Step-7: Compile and Run the make file using commands,

make -f makefilesa

make -f makefilesa run

Step-8: Stop the process.

**Multiple Files:**

STEP-1:Start the process.

STEP-2: Create a file to print the multiplication table for the user entered number.

STEP-3: Convert the single file to multiple files by converting it as a function.

STEP-4: Main function as main.c and function definition as multiple.c and the function protocol in multiple.h file.

STEP-5: Create a make file for those multiple files.

STEP-6: Enter the command to compile those files,

makefile:app.c mul.c

gcc app.c mul.c -o multi

STEP-7: Enter the command to run the file ,

run:

./multi

STEP-8: Enter the command to remove the output file,

clean:

rm multi

STEP-9: Compile and Run the make file using commands,

make -f makefilea1

make -f makefilea1 run

STEP-10: Stop the process.

**PROGRAM**

**Single File:**

```
#include<stdio.h>
void MultiplicationTable(int n)
{
 for(int i=1;i<=10;++i)
  {
    printf("%d*%d=%d\n",n,i,n*i);
  }
}
int main()
{
 int num;
 printf("Enter the integer:");
```

```
 scanf("%d",&num);
 MultiplicationTable(num);
 return 0;
}
```

**Multiple Files:**

**app.c**

```
#include<stdio.h>
#include"mul.h"
int main() {
    int num;

    printf("Enter an integer: ");
    scanf("%d", &num);

    MultiplicationTable(num);

    return 0;
}
```

**Mul.c**

```
#include <stdio.h>
void MultiplicationTable(int n)
{
    for (int i = 1; i <= 10; ++i)
    {
        printf("%d * %d = %d\n", n, i, n * i);
    }
}
```

**mul.h**

void MultiplicationTable(int n);

## OUTPUT

### Single Files

**Multiple Files**

```
monishak@intellect-192:~/Documents/MCA_Sem_1/23MC101_Embedded_C/Embedded_C_Lab_Prog
rams/exercise_11/multi$ make
gcc -o app app.c mul.c
monishak@intellect-192:~/Documents/MCA_Sem_1/23MC101_Embedded_C/Embedded_C_Lab_Prog
rams/exercise_11/multi$ ./app
Enter an integer: 2
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
2 * 6 = 12
2 * 7 = 14
2 * 8 = 16
2 * 9 = 18
2 * 10 = 20
monishak@intellect-192:~/Documents/MCA_Sem_1/23MC101_Embedded_C/Embedded_C_Lab_Prog
rams/exercise_11/multi$
```

**RESULT**

| Ex.no: 12<br>Date: | COMPLEX IMPLEMENTATION OF MAKEFILE TO COMPLIE MULTIPLE FILES |
|---|---|

**Aim:**

To develop a C program that calculates the area of a rectangle using modular programming and compile it using a Makefile for efficient compilation and management of multiple files.

**Algorithm:**

**Step 1**: Start.
**Step 2**: Create a header file rectangle.h to declare the function for calculating the area of a rectangle.
**Step 3**: Implement the function in rectangle.c to calculate the area of a rectangle given its length and width.
**Step 4**: Create a main.c file to:

Accept the length and width of the rectangle as input from the user.

Use the function to calculate the area and display it.

**Step 5**: Write a Makefile:

Define rules to compile individual source files into object files.
Link object files to create the final executable.
Provide a clean rule to remove generated files.

**Step 6**: Compile the program using make.
**Step 7**: Run the program and provide the required input.
**Step 8**: Display the output.
**Step 9**: End.

**PROGRAM**

1. rectangle.h:

```
#ifndef RECTANGLE_H
#define RECTANGLE_H

int calculate_area(int length, int width);

#endif
```

2. rectangle.c:

```
#include "rectangle.h"
```

```
int calculate_area(int length, int width) {
    return length * width;
}
```

3. main.c:

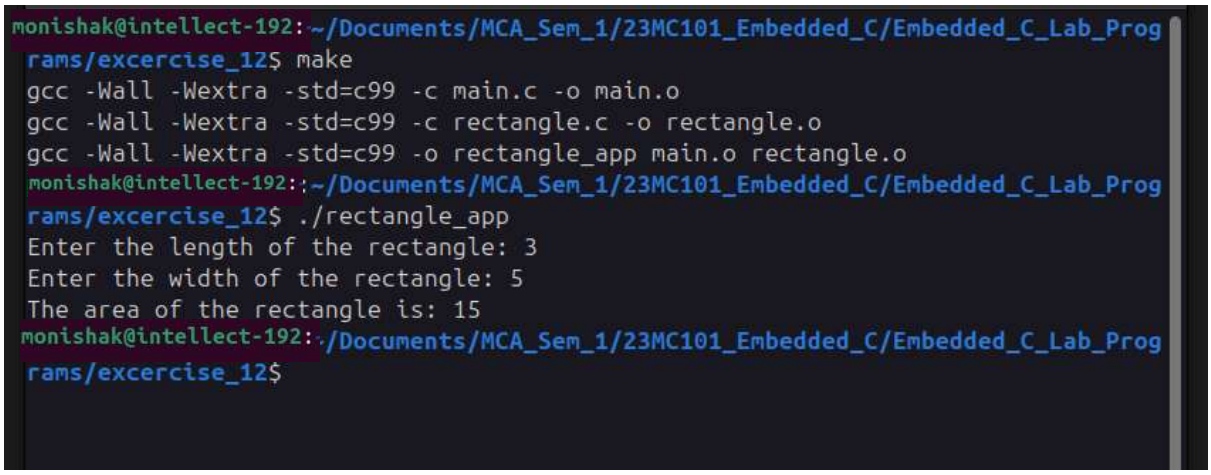
```
#include <stdio.h>
#include "rectangle.h"

int main() {
    int length, width;

    printf("Enter the length of the rectangle: ");
    scanf("%d", &length);

    printf("Enter the width of the rectangle: ");
    scanf("%d", &width);

    int area = calculate_area(length, width);
    printf("The area of the rectangle is: %d\n", area);

    return 0;
}
```

4. Makefile:

```
# Compiler
CC = gcc

# Compiler flags
CFLAGS = -Wall -Wextra -std=c99

# Target executable
TARGET = rectangle_app

# Source files
SRCS = main.c rectangle.c

# Header files
HEADERS = rectangle.h

# Object files
OBJS = $(SRCS:.c=.o)

# Default target
all: $(TARGET)

# Build the target executable
```

```
$(TARGET): $(OBJS)
    $(CC) $(CFLAGS) -o $(TARGET) $(OBJS)

# Rule to build object files
%.o: %.c $(HEADERS)
    $(CC) $(CFLAGS) -c $< -o $@

# Clean up generated files
clean:
    rm -f $(TARGET) $(OBJS)

# Phony targets
.PHONY: all clean
```

**OUTPUT:**




**RESULT**

| Ex.no: 13<br>Date: | CREATING STATIC LIBRARY AND USE THE LIBRARY IN C PROGRAM |
|---|---|

**AIM**

To create a static library for performing arithmetic operations and use the library in a C program.

**ALGORITHM**

**Step 1**: Start.
**Step 2**: Write a header file arith.h that declares functions for addition and subtraction.
**Step 3**: Implement the functions in arith.c to perform addition and subtraction.
**Step 4**: Write the main program in main.c to:
a) Include the header file arith.h.
b) Call the functions defined in the library to perform addition and subtraction.
**Step 5**: Compile arith.c into an object file using the gcc -c command.
**Step 6**: Create a static library libarith.a using the ar command.
**Step 7**: Compile main.c and link it with the static library.
**Step 8**: Run the program to verify the output.
**Step 9**: End.

**PROGRAM**

1. arith.h:

```
#ifndef ARITH_H

#define ARITH_H

int add(int a, int b);

int subtract(int a, int b);

#endif
```

2.arith.c:

```
#include "arith.h"

int add(int a, int b) {

   return a + b;

}
```

```
int subtract(int a, int b) {
    return a - b;
}
```

3. main.c:

```
#include <stdio.h>
#include "arith.h"
int main() {
    int num1 = 15, num2 = 7;
    printf("Addition: %d + %d = %d\n", num1, num2, add(num1, num2));
    printf("Subtraction: %d - %d = %d\n", num1, num2, subtract(num1, num2));
    return 0;
}
```

**Steps to Execute:**

1. **Compile the Library Source File**:

```
gcc -c arith.c -o arith.o
```

2. **Create the Static Library**:

```
ar rcs libarith.a arith.o
```

3. **Compile the Main Program with the Static Library**:

```
gcc main.c -L. -larith -o main
```

4. **Run the Program**:

```
./main
```

**OUTPUT**


**RESULT**

| Ex.no: 14<br>Date: | CREATING DYNAMIC LIBRARY AND USE THE LIBRARY IN C PROGRAM |
|---|---|

## AIM

To create a dynamic library for string operations and use it in a C program.

## ALGORITHM

**Step 1**: Start.
**Step 2**: Write a header file string_ops.h to declare functions for reversing and converting a string to uppercase.
**Step 3**: Implement the functions in string_ops.c for string reversal and uppercase conversion.
**Step 4**: Compile string_ops.c into a position-independent object file using the -fPIC option.
**Step 5**: Create the dynamic library libstring_ops.so using the gcc -shared command.
**Step 6**: Write a main program in main.c to:
a) Include the header file string_ops.h.
b) Call the functions defined in the library to reverse and convert a string to uppercase.
**Step 7**: Compile main.c and link it with the dynamic library using the -L and -l options.
**Step 8**: Set the LD_LIBRARY_PATH to the directory containing the dynamic library.
**Step 9**: Run the program to verify the output.
**Step 10**: End.

## PROGRAM

**Code**

1. string_ops.h (Header File):

```
#ifndef STRING_OPS_H

#define STRING_OPS_H

void reverseString(char* str);

void toUpperCase(char* str);

#endif
```

2. string_ops.c (Source File for Library):

```
 #include <stdio.h>

#include <ctype.h>

#include <string.h>

#include "string_ops.h"

void reverseString(char* str) {
```

```
    int len = strlen(str);
    for (int i = 0; i < len / 2; ++i) {
        char temp = str[i];
        str[i] = str[len - 1 - i];
        str[len - 1 - i] = temp;
    }
}
void toUpperCase(char* str) {
    for (int i = 0; str[i] != '\0'; ++i) {
        str[i] = toupper(str[i]);
    }
}
```

3. main.c (Main Program File):

```
 #include <stdio.h>
#include "string_ops.h"
int main() {
    char str1[100] = "hello";
    char str2[100] = "world";
    printf("Original String 1: %s\n", str1);
    reverseString(str1);
    printf("Reversed String 1: %s\n", str1);
    printf("Original String 2: %s\n", str2);
    toUpperCase(str2);
    printf("Uppercase String 2: %s\n", str2);
    return 0;
}
```

**Steps to Execute**

**Step 1: Compile the Library Source Code**

1. Compile string_ops.c into a position-independent object file:

```
gcc -c -fPIC string_ops.c -o string_ops.o
```

**Step 2: Create the Dynamic Library**

1. Use `gcc` with the `-shared` option to create the dynamic library:

```
gcc -shared -o libstring_ops.so string_ops.o
```

**Step 3: Compile the Main Program**

1. Compile `main.c` and link it with the dynamic library:

```
gcc main.c -L. -lstring_ops -o main
```

**Step 4: Set the Library Path**

1. Set the `LD_LIBRARY_PATH` environment variable to include the current directory:

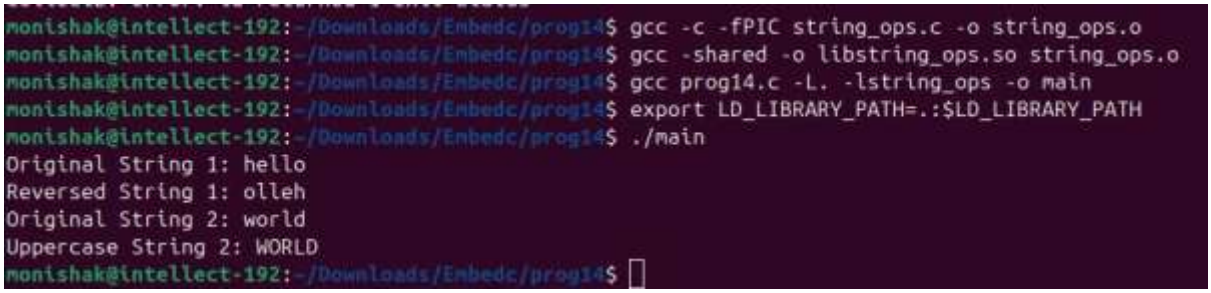
```
export LD_LIBRARY_PATH=.:$LD_LIBRARY_PATH
```

**Step 5: Run the Program**

1. Execute the program:

```
./main
```

**OUTPUT**




**RESULT**